10-601 Introduction to Machine Learning

Machine Learning Department
School of Computer Science
Carnegie Mellon University

RNNs

+

PAC Learning

Matt Gormley
Lecture 15
Mar. 6, 2019

Reminders

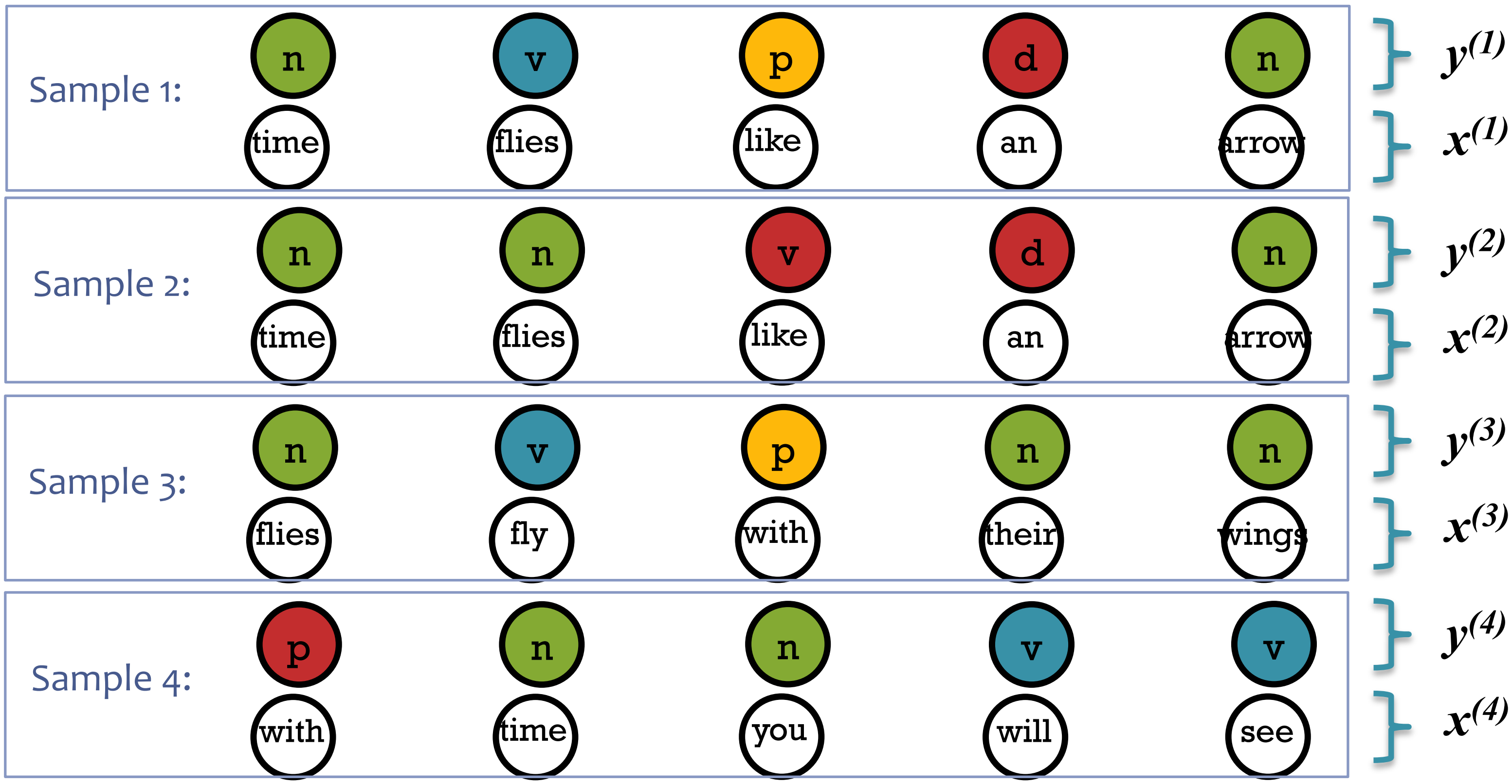
- **Homework 5: Neural Networks**
  - Out: Fri, Mar 1
  - Due: Fri, Mar 22 at 11:59pm
- **Today's In-Class Poll**
  - http://p15.mlcourse.org

Q&A

RECURRENT NEURAL NETWORKS

Dataset for Supervised Part-of-Speech (POS) Tagging

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



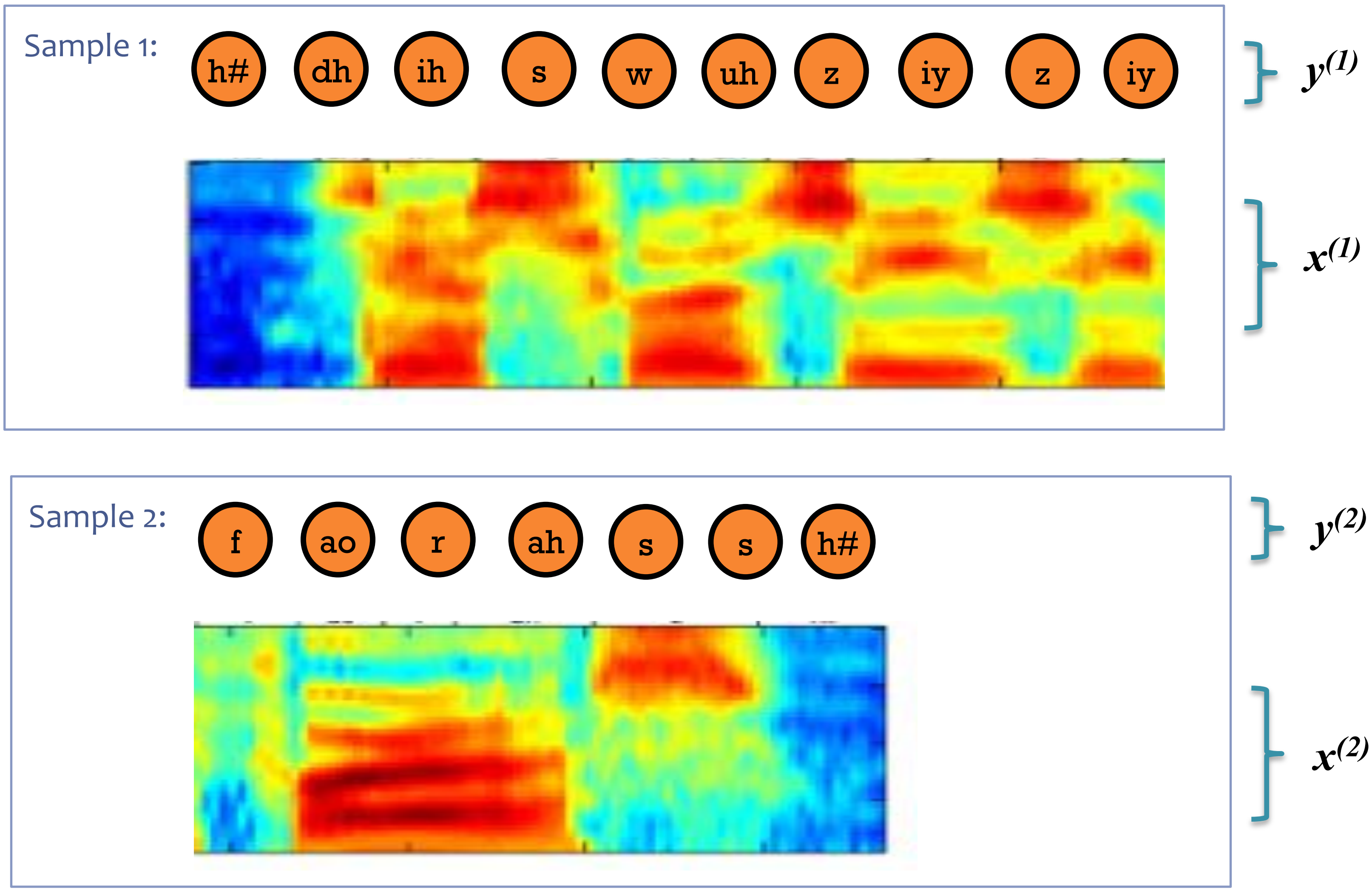
Dataset for Supervised Handwriting Recognition

Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$

Figures from (Chatzis & Demiris, 2013)

Dataset for Supervised Phoneme (Speech) Recognition

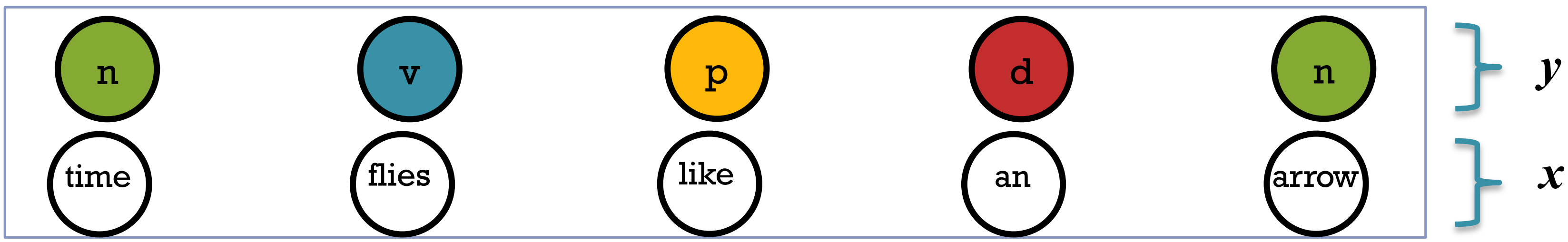
Data: $\mathcal{D} = \{\boldsymbol{x}^{(n)}, \boldsymbol{y}^{(n)}\}_{n=1}^{N}$



Figures from (Jansen & Niyogi, 2013)

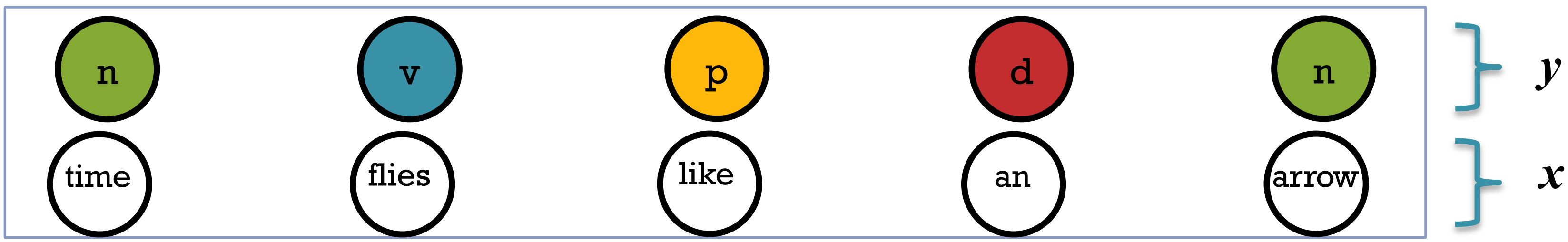
Time Series Data

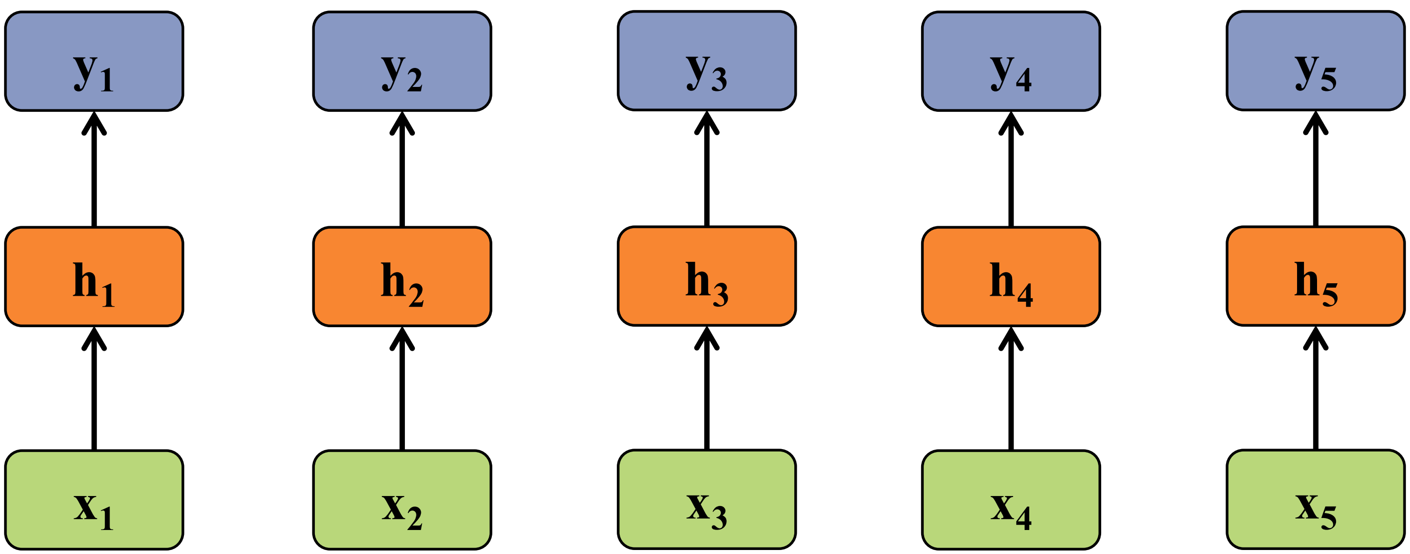
Question 1: How could we apply the neural networks we've seen so far (which expect **fixed size input/output**) to a prediction task with **variable length input/output**?



Time Series Data

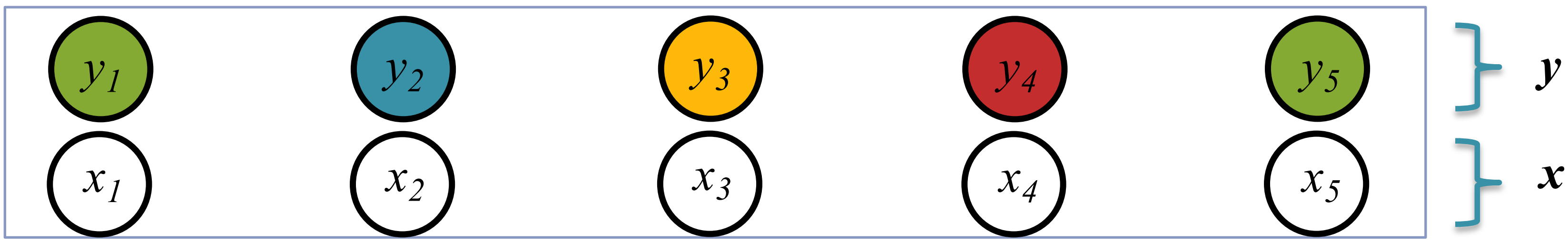
Question 1: How could we apply the neural networks we've seen so far (which expect **fixed size input/output**) to a prediction task with **variable length input/output**?





Time Series Data

Question 2: How could we incorporate context (e.g. words to the left/right, or tags to the left/right) into our solution?



Multiple Choice:

Working left-to-right, use features of…

	x_{i-1}	x_i	x_{i+1}	y_{i-1}	y_i	y_{i+1}
A	✓					
B				✓		
C	✓			✓		
D	✓			✓	✓	✓
E	✓	✓		✓	✓	✓
F	✓	✓	✓	✓		
G	✓	✓	✓	✓	✓	
H	✓	✓	✓	✓	✓	✓

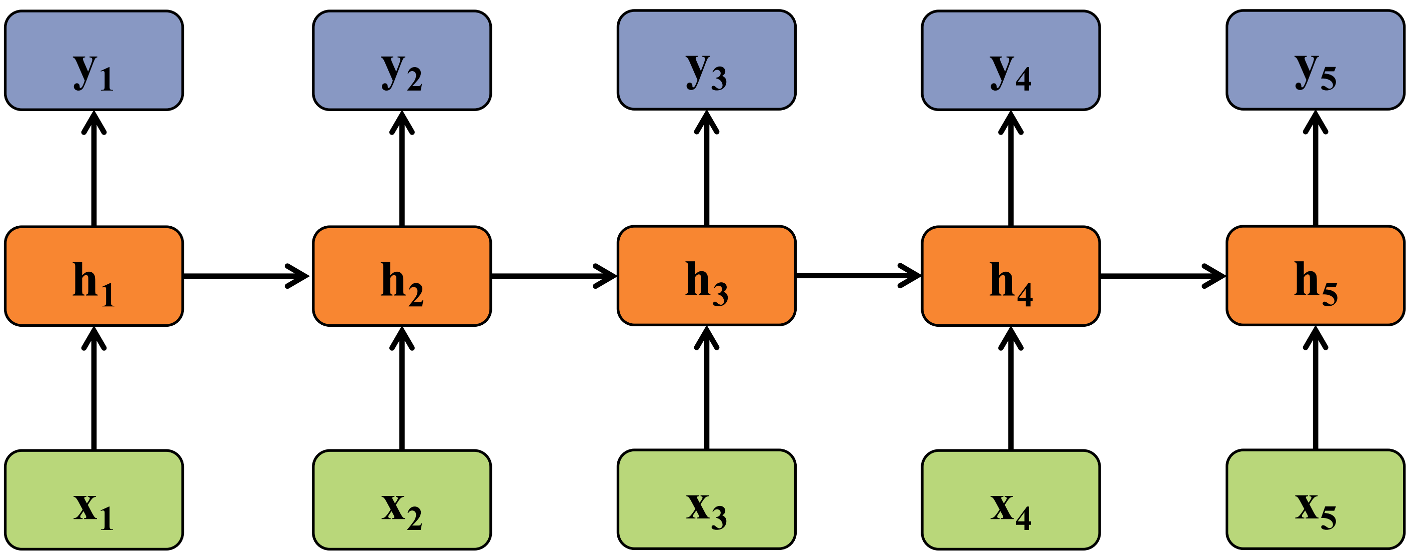
Recurrent Neural Networks (RNNs)

$$\begin{aligned}
\text{inputs: } & \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I \\
\text{hidden units: } & \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J \\
\text{outputs: } & \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K \\
\text{nonlinearity: } & \mathcal{H}
\end{aligned}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$

$$y_t = W_{hy} h_t + b_y$$



Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

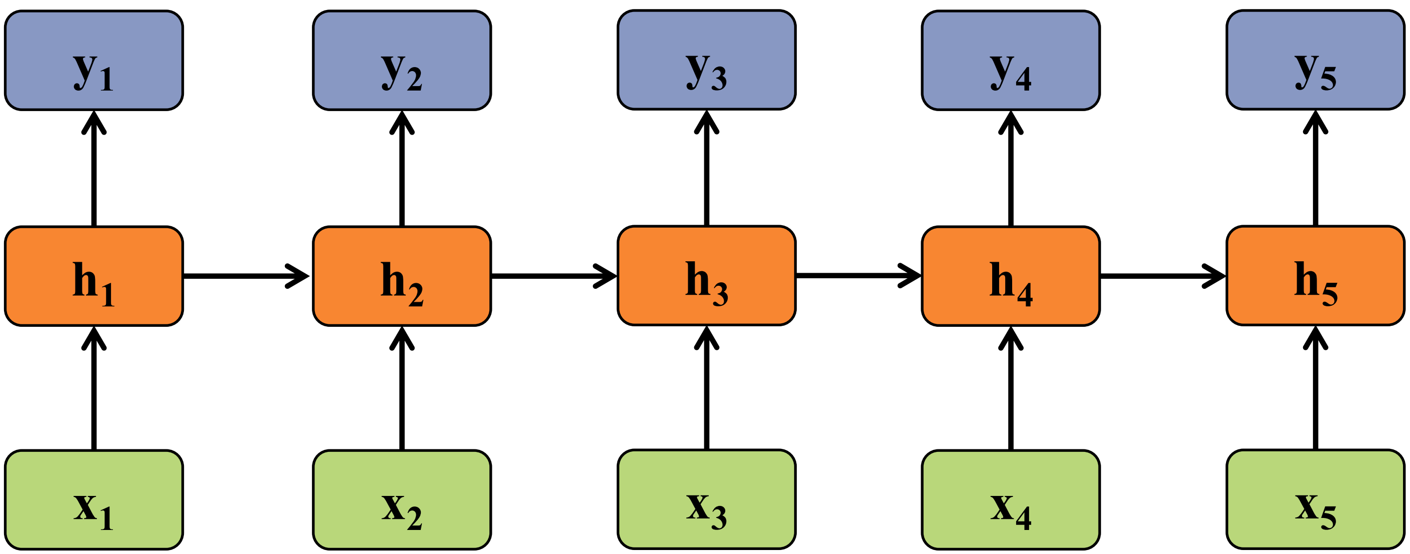
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$

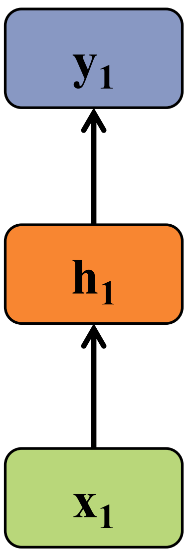


Recurrent Neural Networks (RNNs)

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh} x_t + W_{hh} h_{t-1} + b_h\right)$$
$$y_t = W_{hy} h_t + b_y$$



- If $T=1$, then we have a standard feed-forward **neural net with one hidden layer**
- All of the deep nets from last lecture required **fixed size inputs/outputs**

Background

A Recipe for Machine Learning

1. Given training data:

$$\{\boldsymbol{x}_i, \boldsymbol{y}_i\}_{i=1}^N$$

2. Choose each of these:
   - Decision function

$$\hat{\boldsymbol{y}} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

   - Loss function

$$\ell(\hat{\boldsymbol{y}}, \boldsymbol{y}_i) \in \mathbb{R}$$

3. Define goal:

$$\boldsymbol{\theta}^* = \arg\min_{\boldsymbol{\theta}} \sum_{i=1}^N \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

4. Train with SGD:
(take small steps opposite the gradient)

$$\boldsymbol{\theta}^{(t+1)} = \boldsymbol{\theta}^{(t)} - \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Background

A Recipe for Machine Learning

1.

- Recurrent Neural Networks (RNNs) provide another form of **decision function**
- An RNN is just another differential function

$\boldsymbol{y}_i)$

2. Choose each of these:

– Decision function

$$\hat{y} = f_{\boldsymbol{\theta}}(\boldsymbol{x}_i)$$

4. Train with SGD:

(take small steps opposite the gradient)

- We'll just need a method of computing the gradient efficiently
- Let's use Backpropagation Through Time...

$$- \eta_t \nabla \ell(f_{\boldsymbol{\theta}}(\boldsymbol{x}_i), \boldsymbol{y}_i)$$

Recurrent Neural Networks (RNNs)

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$

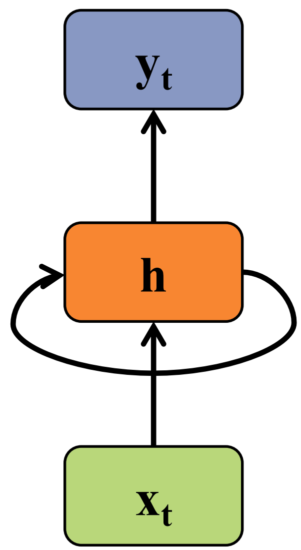
outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

nonlinearity: $\mathcal{H}$

Definition of the RNN:

$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$

$$y_t = W_{hy}h_t + b_y$$



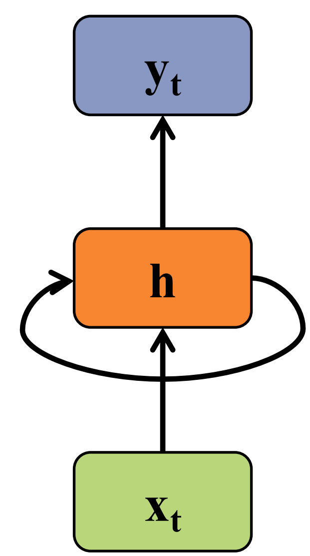
Recurrent Neural Networks (RNNs)

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \mathbf{h} = (h_1, h_2, \ldots, h_T), h_i \in \mathcal{R}^J$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Definition of the RNN:

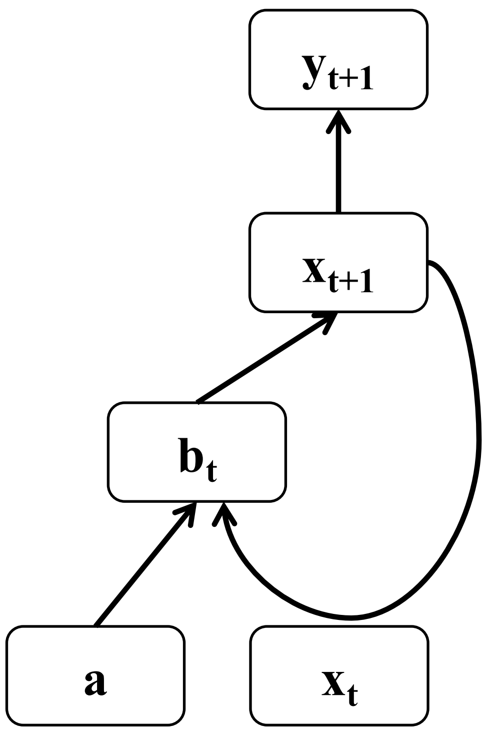
$$h_t = \mathcal{H}\left(W_{xh}x_t + W_{hh}h_{t-1} + b_h\right)$$
$$y_t = W_{hy}h_t + b_y$$

- By unrolling the RNN through time, we can **share parameters** and accommodate **arbitrary length** input/output pairs
- Applications: **time-series data** such as sentences, speech, stock-market, signal data, etc.



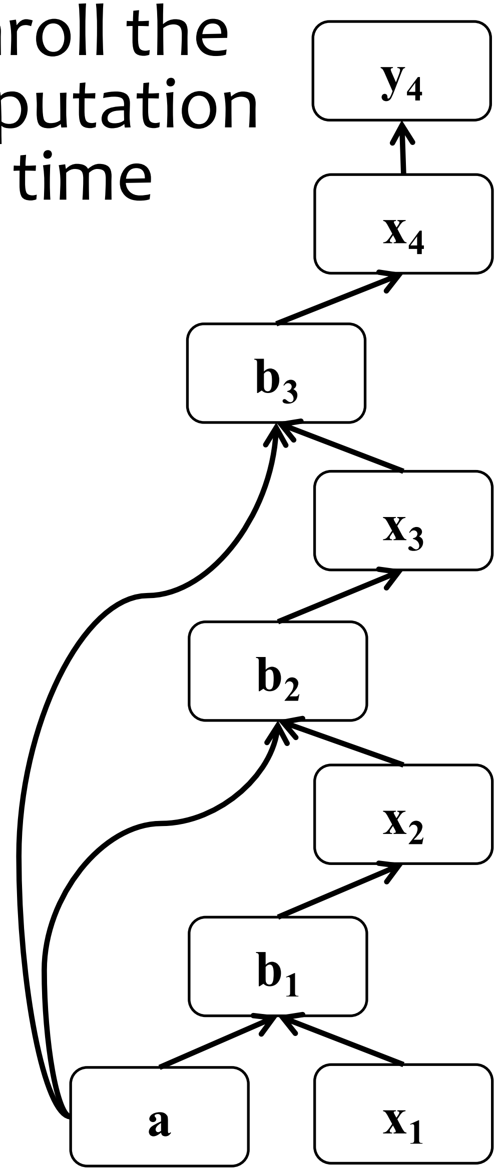
Background: Backprop through time

Recurrent neural network:



BPTT:

1. Unroll the computation over time



2. Run backprop through the resulting feed-forward network

(Robinson & Fallside, 1987)
(Werbos, 1988)
(Mozer, 1995)

Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$

$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$

$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$

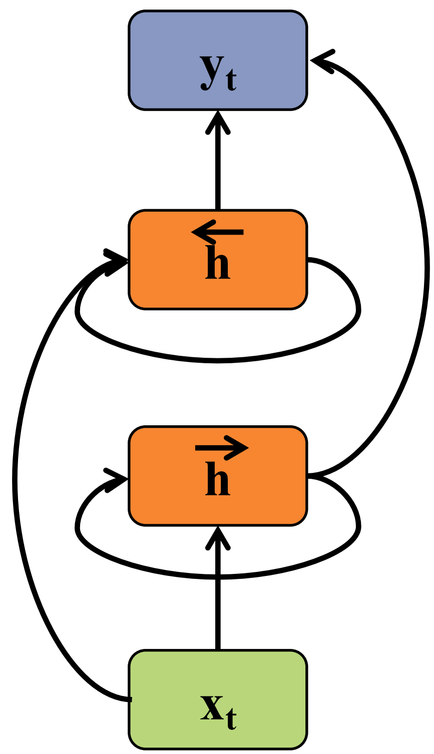
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$

$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$

$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$

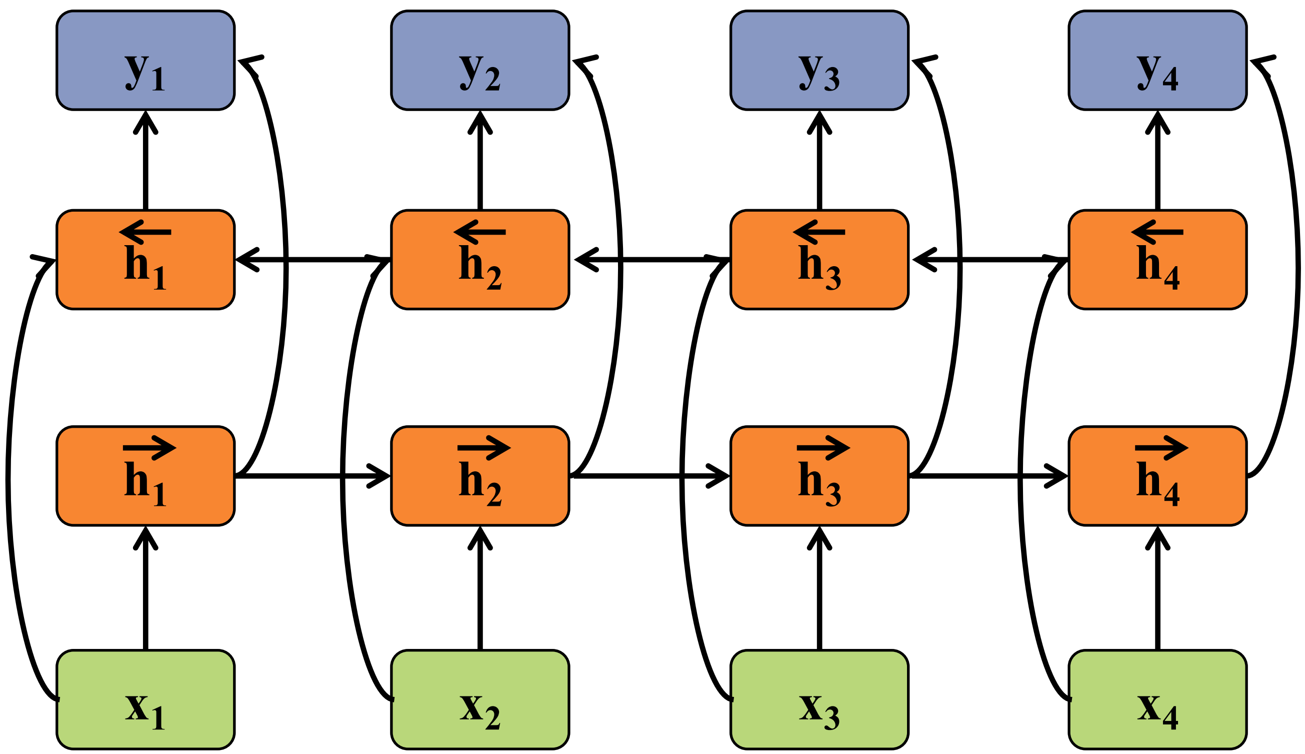
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Bidirectional RNN

inputs: $\mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$

hidden units: $\overrightarrow{\mathbf{h}}$ and $\overleftarrow{\mathbf{h}}$

outputs: $\mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$

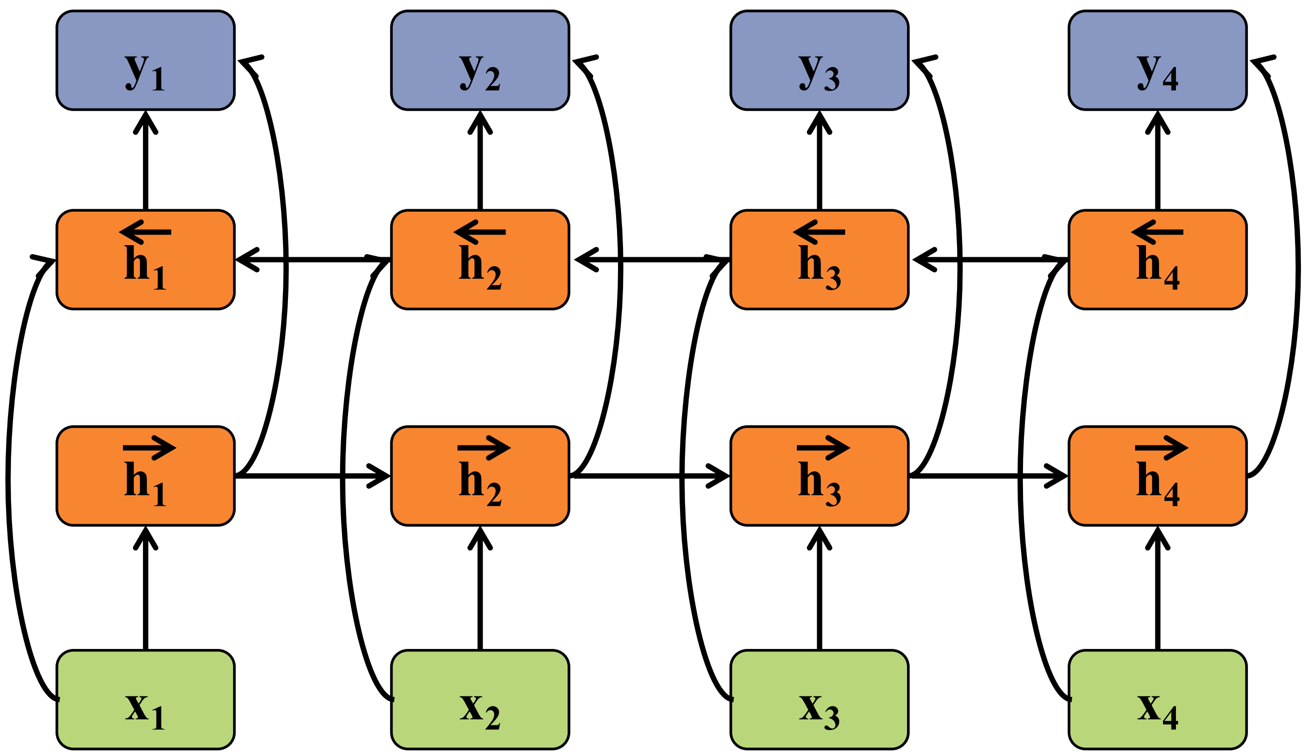
nonlinearity: $\mathcal{H}$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$

$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$

$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$

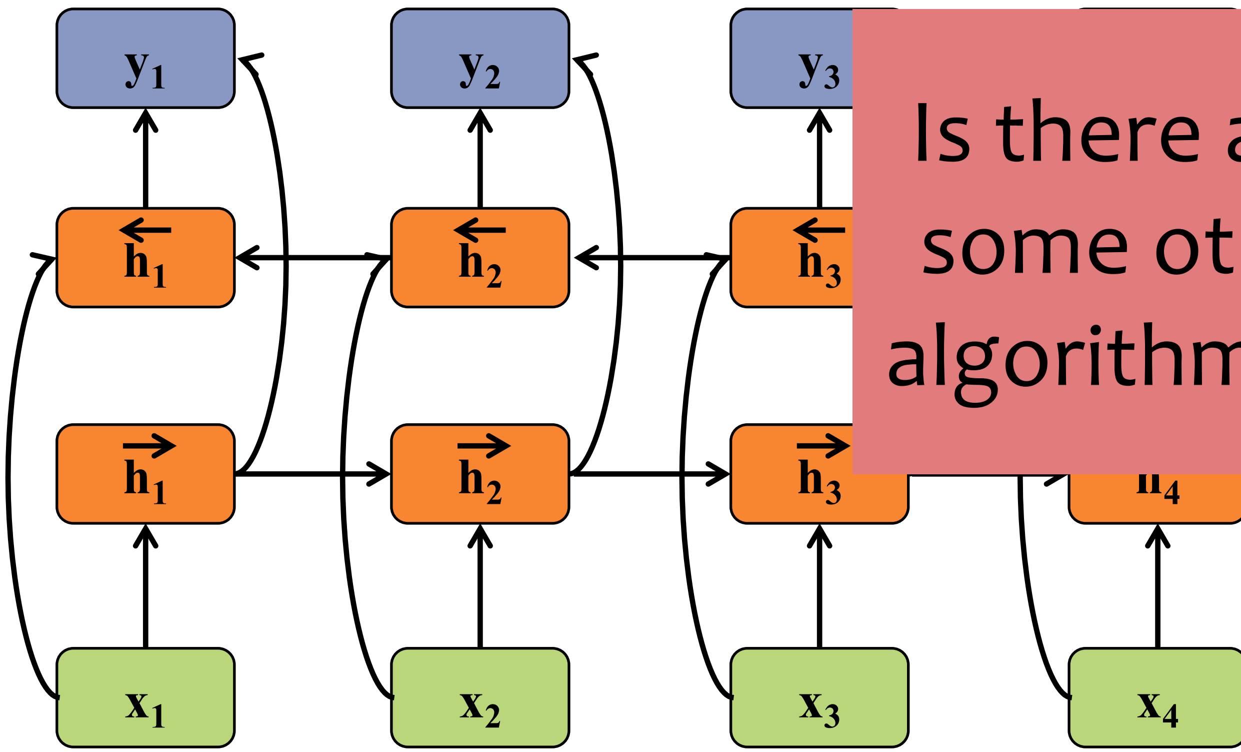


Bidirectional RNN

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{hidden units: } \overrightarrow{\mathbf{h}} \text{ and } \overleftarrow{\mathbf{h}}$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$\overrightarrow{h}_t = \mathcal{H}\left(W_{x\overrightarrow{h}} x_t + W_{\overrightarrow{h}\overrightarrow{h}} \overrightarrow{h}_{t-1} + b_{\overrightarrow{h}}\right)$$
$$\overleftarrow{h}_t = \mathcal{H}\left(W_{x\overleftarrow{h}} x_t + W_{\overleftarrow{h}\overleftarrow{h}} \overleftarrow{h}_{t+1} + b_{\overleftarrow{h}}\right)$$
$$y_t = W_{\overrightarrow{h}y} \overrightarrow{h}_t + W_{\overleftarrow{h}y} \overleftarrow{h}_t + b_y$$



Is there an analogy to some other recursive algorithm(s) we know?

Deep RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$

$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$

$$\text{nonlinearity: } \mathcal{H}$$

Recursive Definition:

$$h_t^n = \mathcal{H}\left(W_{h^{n-1}h^n} h_t^{n-1} + W_{h^n h^n} h_{t-1}^n + b_h^n\right)$$

$$y_t = W_{h^N y} h_t^N + b_y$$

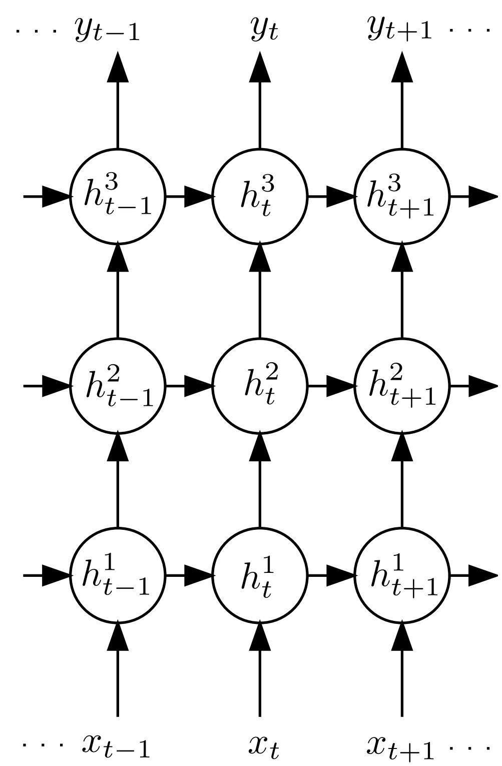


Figure from (Graves et al., 2013)

Deep Bidirectional RNNs

$$\text{inputs: } \mathbf{x} = (x_1, x_2, \ldots, x_T), x_i \in \mathcal{R}^I$$
$$\text{outputs: } \mathbf{y} = (y_1, y_2, \ldots, y_T), y_i \in \mathcal{R}^K$$
$$\text{nonlinearity: } \mathcal{H}$$

- Notice that the upper level hidden units have input from **two previous layers** (i.e. wider input)
- Likewise for the output layer
- What analogy can we draw to DNNs, DBNs, DBMs?

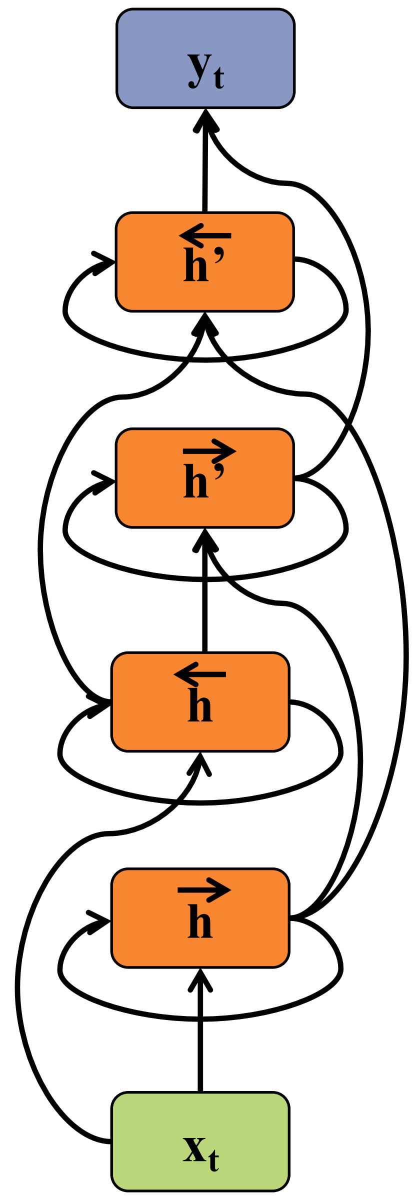
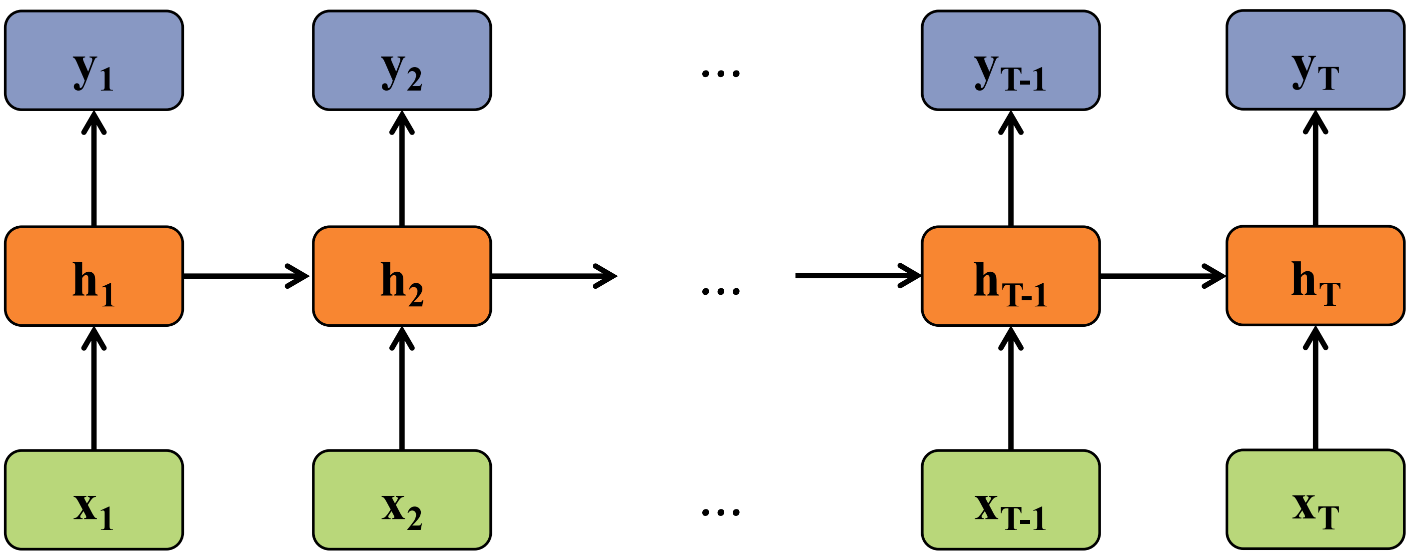


Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)

Motivation:

- Standard RNNs have trouble learning long distance dependencies
- LSTMs combat this issue



Long Short-Term Memory (LSTM)

Motivation:

- Vanishing gradient problem for Standard RNNs
- Figure shows sensitivity (darker = more sensitive) to the input at time t=1

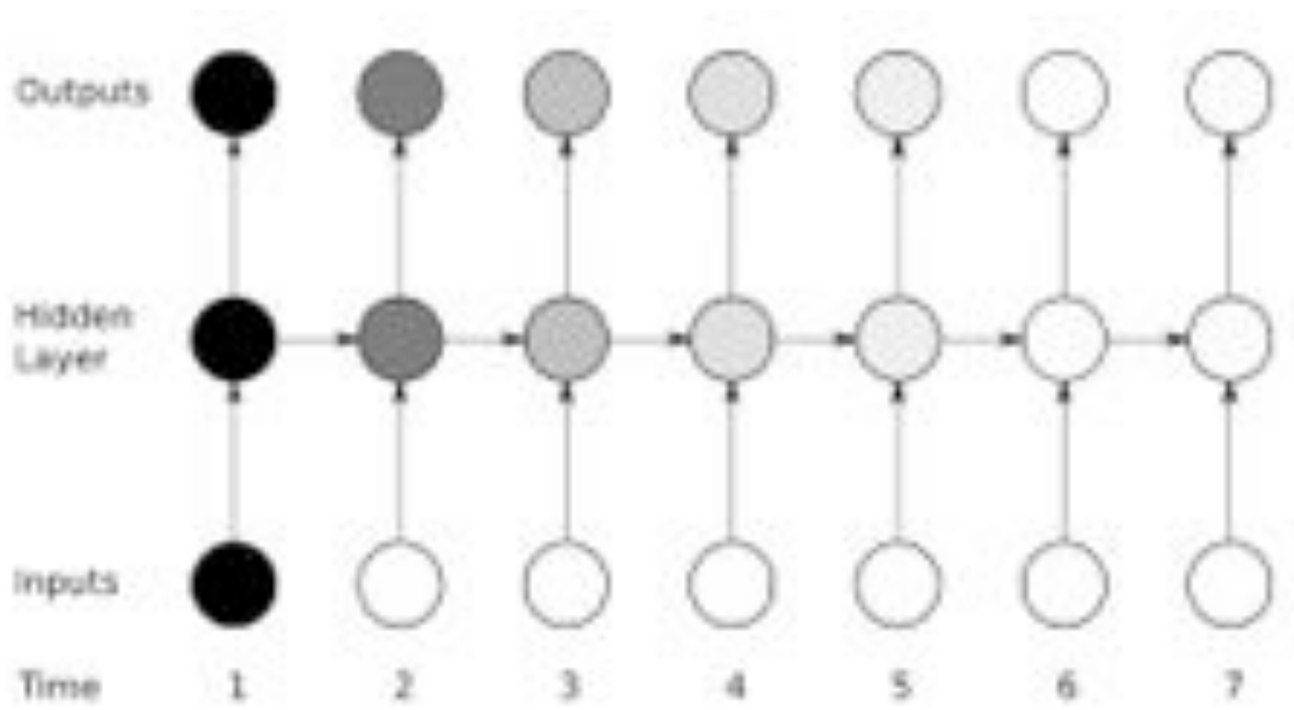


Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)

Motivation:

- LSTM units have a rich internal structure
- The various "gates" determine the propagation of information and can choose to "remember" or "forget" information

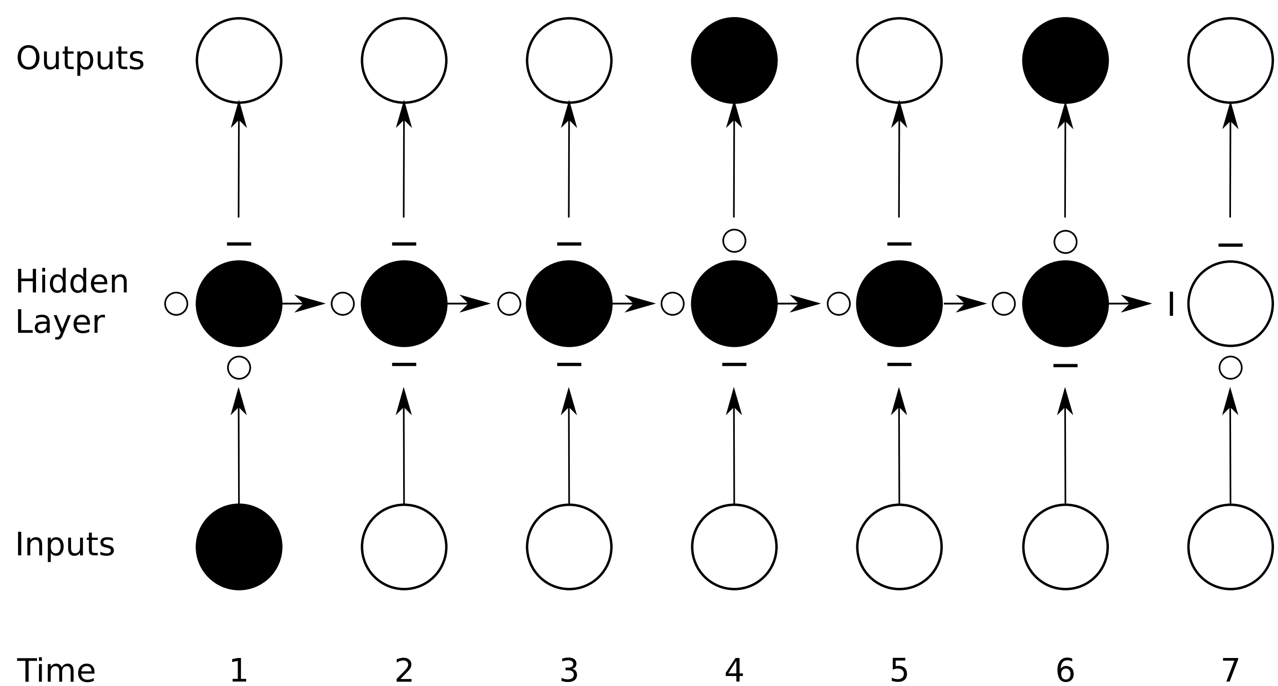
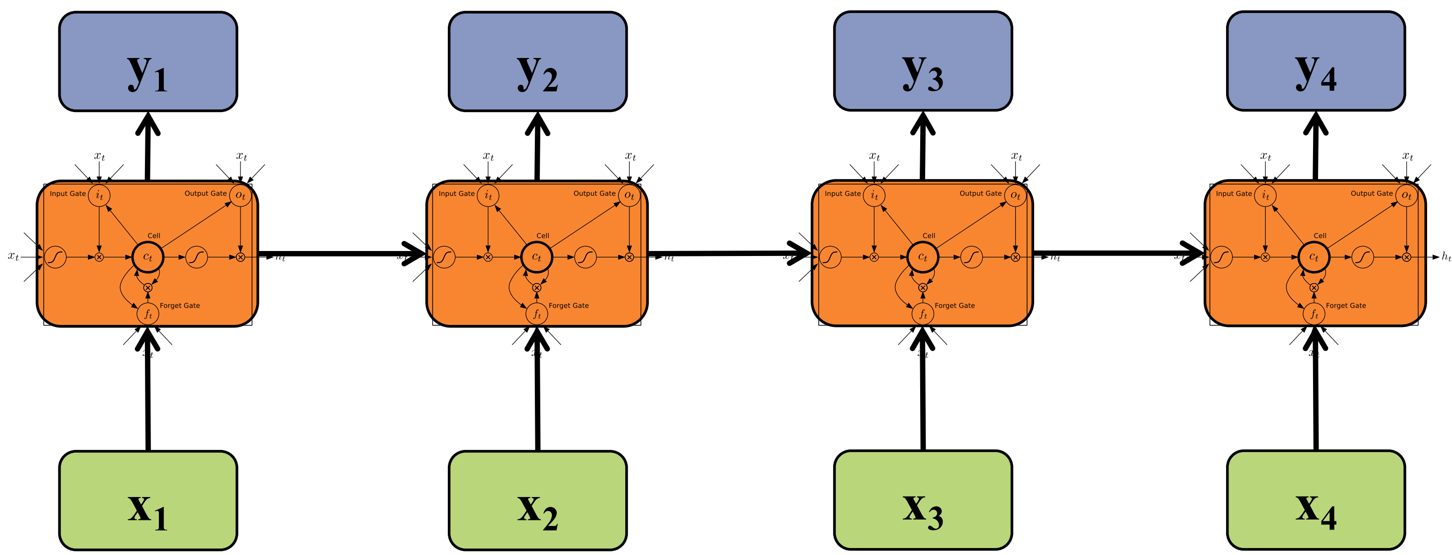


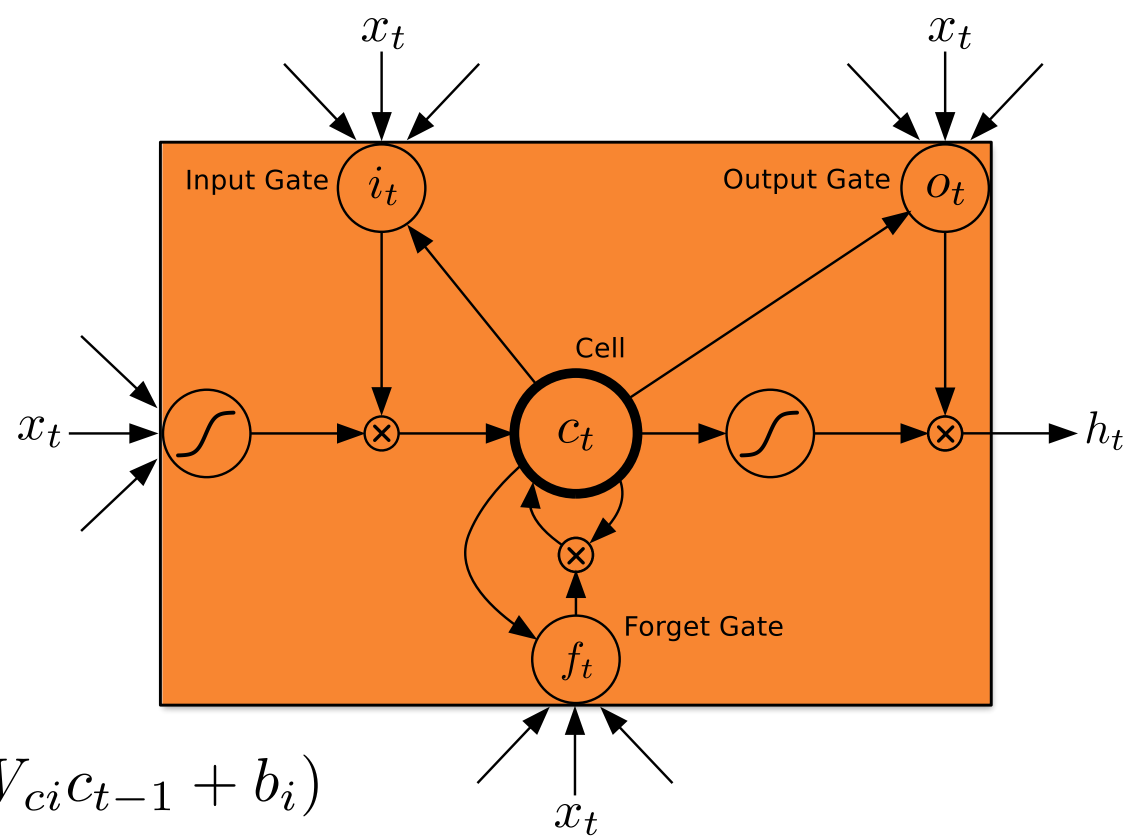
Figure from (Graves, 2012)

Long Short-Term Memory (LSTM)



Long Short-Term Memory (LSTM)

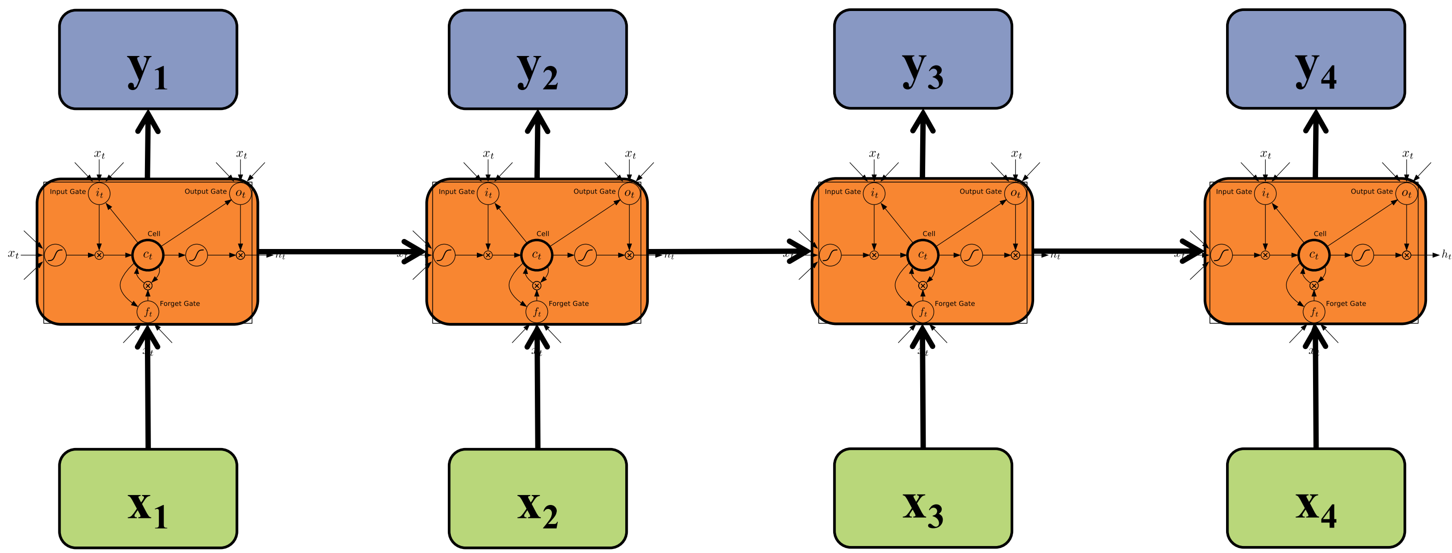
- **Input gate:** masks out the standard RNN inputs
- **Forget gate:** masks out the previous cell
- **Cell:** stores the input/forget mixture
- **Output gate:** masks out the values of the next hidden



$$i_t = \sigma\left(W_{xi}x_t + W_{hi}h_{t-1} + W_{ci}c_{t-1} + b_i\right)$$
$$f_t = \sigma\left(W_{xf}x_t + W_{hf}h_{t-1} + W_{cf}c_{t-1} + b_f\right)$$
$$c_t = f_t c_{t-1} + i_t \tanh\left(W_{xc}x_t + W_{hc}h_{t-1} + b_c\right)$$
$$o_t = \sigma\left(W_{xo}x_t + W_{ho}h_{t-1} + W_{co}c_t + b_o\right)$$
$$h_t = o_t \tanh(c_t)$$

Figure from (Graves et al., 2013)

Long Short-Term Memory (LSTM)



Deep Bidirectional LSTM (DBLSTM)

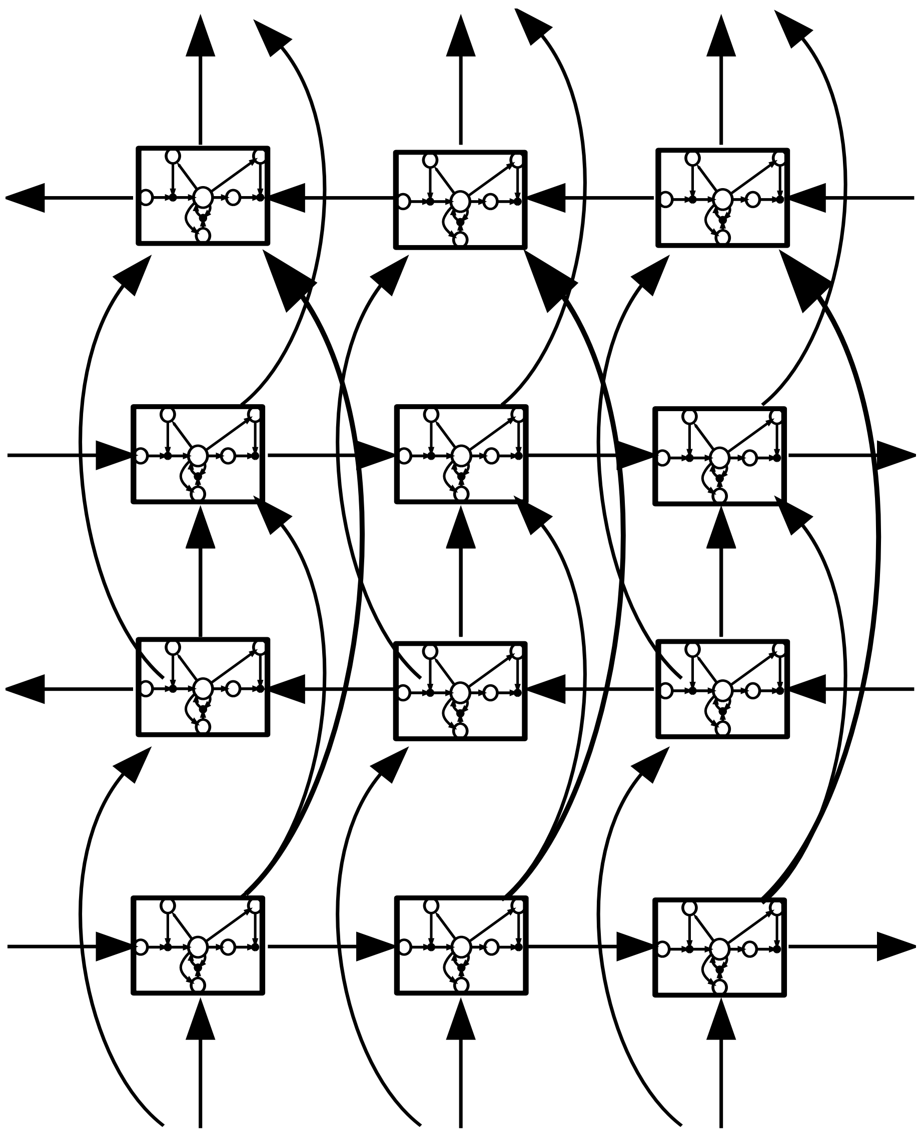


Figure from (Graves et al., 2013)

- Figure: input/output layers not shown
- **Same general topology** as a Deep Bidirectional RNN, but with **LSTM units** in the hidden layers
- No additional **representational power** over DBRNN, but **easier to learn** in practice

Deep Bidirectional LSTM (DBLSTM)

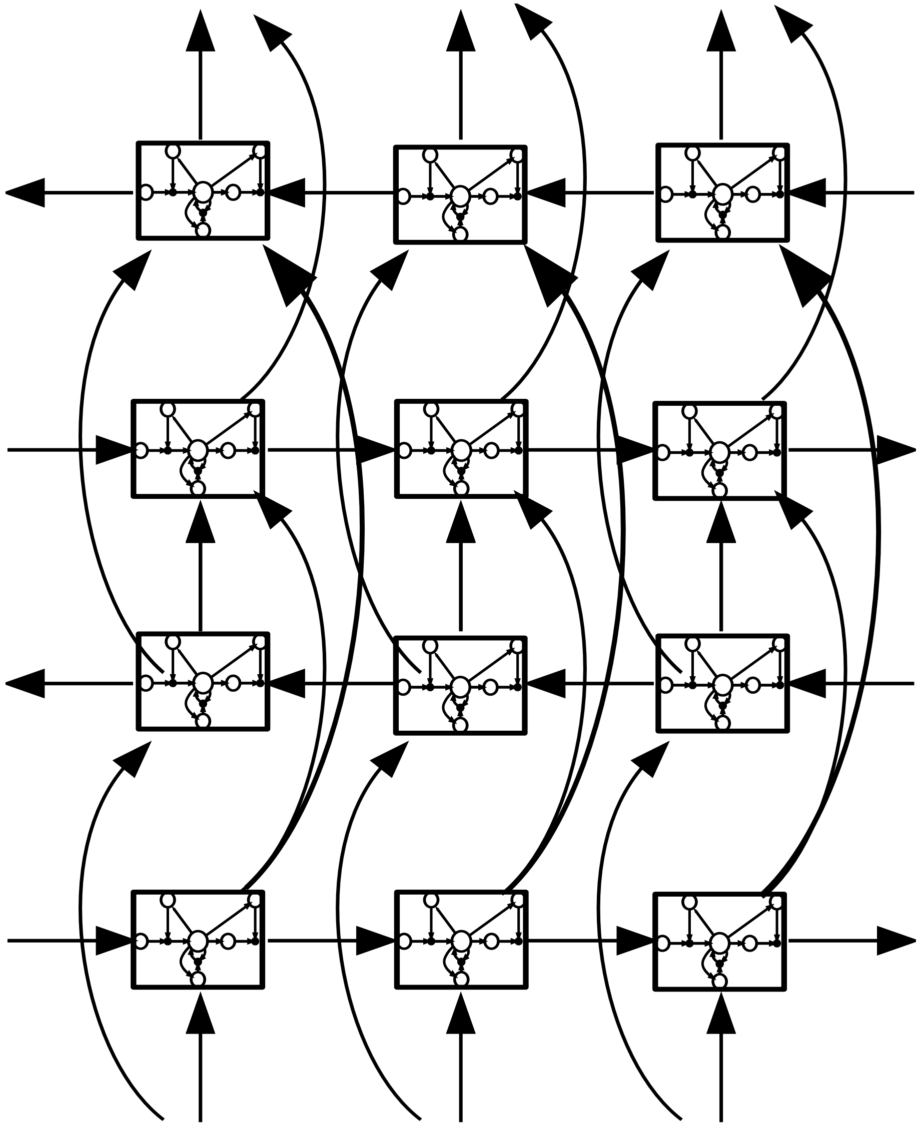


Figure from (Graves et al., 2013)

How important is this particular architecture?

Jozefowicz et al. (2015) **evaluated 10,000 different LSTM-like architectures** and found several variants that worked just as well on several tasks.

RNN Summary

- **RNNs**
  - Applicable to tasks such as **sequence labeling**, speech recognition, machine translation, etc.
  - Able to **learn context features** for time series data
  - Vanishing gradients are still a problem – but **LSTM units** can help
- **Other Resources**
  - Christopher Olah's blog post on LSTMs http://colah.github.io/posts/2015-08-Understanding-LSTMs/

LEARNING THEORY

PAC-MAN Learning

For some hypothesis $h \in \mathcal{H}$:

1. True Error

$$R(h)$$

2. Training Error

$$\hat{R}(h)$$

Question 2:
What is the expected number of PAC-MAN levels Matt will complete before a **Game-Over**?

A. 1-10
B. 11-20
C. 21-30

ML Big Picture

Learning Paradigms:

What data is available and when? What form of prediction?

- supervised learning
- unsupervised learning
- semi-supervised learning
- reinforcement learning
- active learning
- imitation learning
- domain adaptation
- online learning
- density estimation
- recommender systems
- feature learning
- manifold learning
- dimensionality reduction
- ensemble learning
- distant supervision
- hyperparameter optimization

Problem Formulation:

What is the structure of our output prediction?

boolean	Binary Classification
categorical	Multiclass Classification
ordinal	Ordinal Classification
real	Regression
ordering	Ranking
multiple discrete	Structured Prediction
multiple continuous	(e.g. dynamical systems)
both discrete & cont.	(e.g. mixed graphical models)

Application Areas

Key challenges?

NLP, Speech, Computer Vision, Robotics, Medicine, Search

Theoretical Foundations:

What principles guide learning?

- ❑ probabilistic
- ❑ information theoretic
- ❑ evolutionary search
- ❑ ML as optimization

Facets of Building ML Systems:

How to build systems that are robust, efficient, adaptive, effective?

1. *Data prep*
2. *Model selection*
3. *Training (optimization / search)*
4. *Hyperparameter tuning on validation data*
5. *(Blind) Assessment on test data*

Big Ideas in ML:

Which are the ideas driving development of the field?

- *inductive bias*
- *generalization / overfitting*
- *bias-variance decomposition*
- *generative vs. discriminative*
- *deep nets, graphical models*
- *PAC learning*
- *distant rewards*

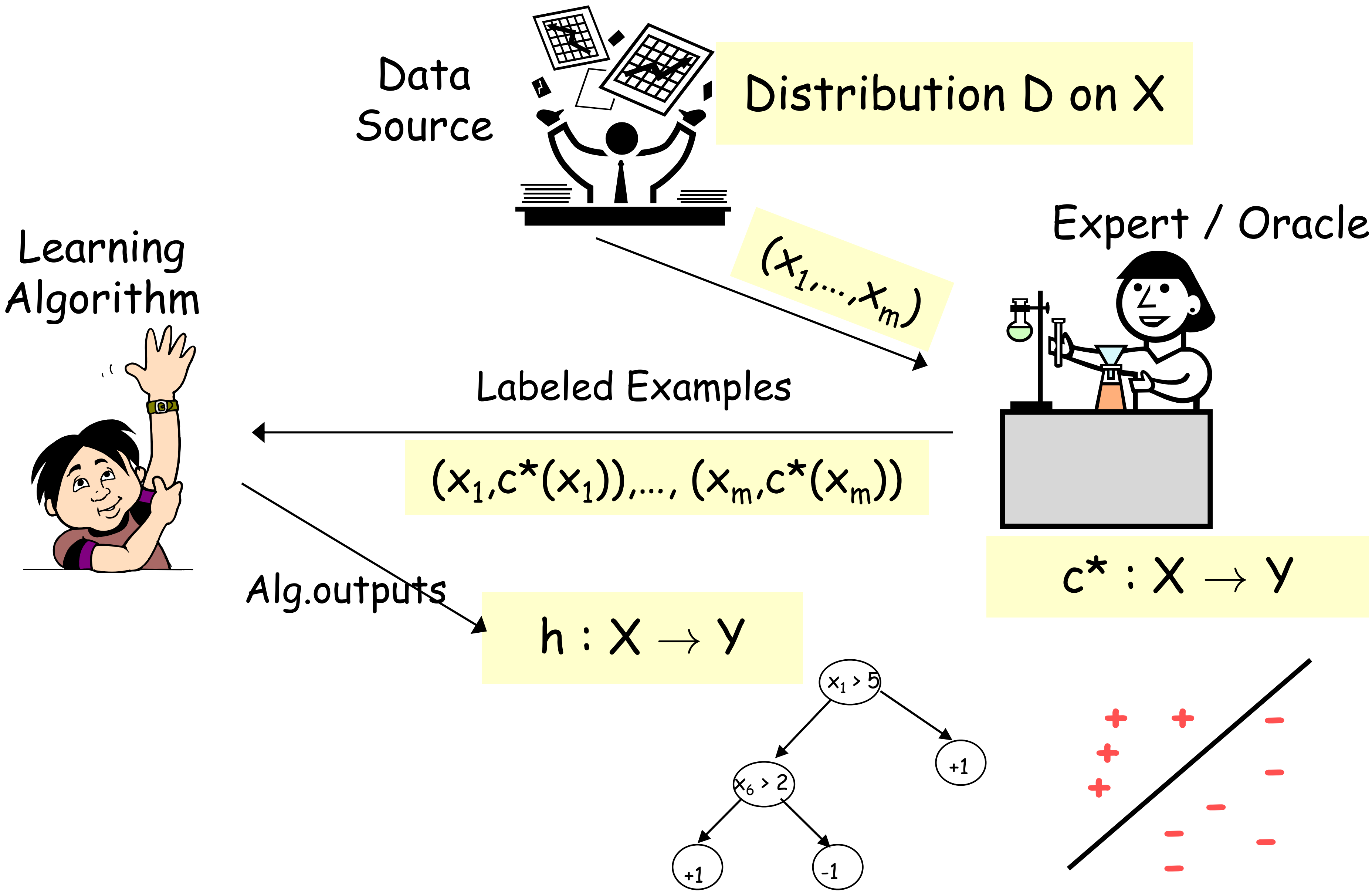
Questions For Today

1. Given a classifier with **zero training error**, what can we say about **true error** (aka. generalization error)?
   (Sample Complexity, Realizable Case)

2. Given a classifier with **low training error**, what can we say about **true error** (aka. generalization error)?
   (Sample Complexity, Agnostic Case)

3. Is there a **theoretical justification for regularization** to avoid overfitting?
   (Structural Risk Minimization)

PAC/SLT models for Supervised Learning



Two Types of Error

1. True Error (aka. **expected risk**)

$$R(h) = P_{\mathbf{x} \sim p^*(\mathbf{x})}(c^*(\mathbf{x}) \neq h(\mathbf{x}))$$

This quantity is always **unknown**

2. Train Error (aka. **empirical risk**)

$$\begin{aligned} \hat{R}(h) &= P_{\mathbf{x} \sim \mathcal{S}}(c^*(\mathbf{x}) \neq h(\mathbf{x})) \\ &= \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(c^*(\mathbf{x}^{(i)}) \neq h(\mathbf{x}^{(i)})) \\ &= \frac{1}{N} \sum_{i=1}^{N} \mathbb{1}(y^{(i)} \neq h(\mathbf{x}^{(i)})) \end{aligned}$$

We can **measure** this on the training data

where $\mathcal{S} = \{\mathbf{x}^{(1)}, \ldots, \mathbf{x}^{(N)})\}_{i=1}^{N}$ is the training data set, and $\mathbf{x} \sim \mathcal{S}$ denotes that $\mathbf{x}$ is sampled from the empirical distribution.

PAC / SLT Model

We've also referred to this as the "Function Approximation View"

1. Generate instances from *unknown* distribution p^*

$$\mathbf{x}^{(i)} \sim p^*(\mathbf{x}), \forall i \tag{1}$$

2. Oracle labels each instance with *unknown* function c^*

$$y^{(i)} = c^*(\mathbf{x}^{(i)}), \forall i \tag{2}$$

3. Learning algorithm chooses hypothesis $h \in \mathcal{H}$ with low(est) training error, $\hat{R}(h)$

$$\hat{h} = \underset{h}{\text{argmin}}\, \hat{R}(h) \tag{3}$$

4. Goal: Choose an h with low generalization error $R(h)$

Three Hypotheses of Interest

The **true function** c^* is the one we are trying to learn and that labeled the training data:

$$y^{(i)} = c^*(\mathbf{x}^{(i)}), \, \forall i \tag{1}$$

The **expected risk minimizer** has lowest true error:

$$h^* = \operatorname*{argmin}_{h \in \mathcal{H}} R(h)$$

Question:
True or False:
h* and c* are always equal.

The **empirical risk minimizer** has lowest training error:

$$\hat{h} = \operatorname*{argmin}_{h \in \mathcal{H}} \hat{R}(h) \tag{3}$$

PAC LEARNING

Probably Approximately Correct (PAC) Learning

Whiteboard:

- PAC Criterion
- Meaning of "Probably Approximately Correct"
- Def: PAC Learner
- Sample Complexity
- Consistent Learner

PAC Learning

The **PAC criterion** is that our learner produces a high accuracy learner with high probability:

$$P(|R(h) - \hat{R}(h)| \leq \epsilon) \geq 1 - \delta \tag{1}$$

Suppose we have a learner that produces a hypothesis $h \in \mathcal{H}$ given a sample of N training examples. The algorithm is called **consistent** if for every ϵ and δ, there exists a positive number of training examples N such that for any distribution p^*, we have that:

$$P(|R(h) - \hat{R}(h)| > \epsilon) < \delta \tag{2}$$

The **sample complexity** is the minimum value of N for which this statement holds. If N is finite for some learning algorithm, then $\mathcal{H}$ is said to be **learnable.** If N is a polynomial function of $\frac{1}{\epsilon}$ and $\frac{1}{\delta}$ for some learning algorithm, then $\mathcal{H}$ is said to be **PAC learnable**.

SAMPLE COMPLEXITY RESULTS

Sample Complexity Results

Definition 0.1. The **sample complexity** of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e. close to 1).

Four Cases we care about…

We'll start with the finite case…

	Realizable	Agnostic
Finite $\|\mathcal{H}\|$		
Infinite $\|\mathcal{H}\|$		

Generalization and Overfitting

Whiteboard:

- Realizable vs. Agnostic Cases
- Finite vs. Infinite Hypothesis Spaces
- Theorem 1: Realizable Case, Finite |H|
- Proof of Theorem 1

Sample Complexity Results

Definition 0.1. The **sample complexity** of a learning algorithm is the number of examples required to achieve arbitrarily small error (with respect to the optimal hypothesis) with high probability (i.e. close to 1).

Four Cases we care about…

	Realizable	Agnostic
Finite $\|\mathcal{H}\|$	**Thm. 1** $N \geq \frac{1}{\epsilon}\left[\log(\|\mathcal{H}\|) + \log(\frac{1}{\delta})\right]$ labeled examples are sufficient so that with probability $(1-\delta)$ all $h \in \mathcal{H}$ with $\hat{R}(h) = 0$ have $R(h) \leq \epsilon$.	
Infinite $\|\mathcal{H}\|$		